Will Marringa
CS6015

Assignment 6: Fuzzer Script

The testing oracles I used in this fuzzer were:

- Address Sanitizer: Environment Rule Violations
- Heuristic Oracle
- Consistency Oracle

Address Sanitizer

The violation of programming environment rules is a critical testing Oracle that does not require a great deal of formulation to work. The use of compile-time and run-time error checking will generally reduce the likelihood of undefined program behavior as you build a project. In addition to utilizing code-linting plugins and compiling my program files with error flagging turned on, the use of address sanitizer will catch memory access issues.

Heuristic Oracle

This type of Oracle provides exact results for a given set of test inputs. I generated randomly-sized squares, rectangles, parallelograms, and trapezoids against their respective answer files to check for bugs in classification. In addition to testing valid, randomly-sized shapes, I also tested various errors via randomly generated inputs violating the number of input points, duplicate indices, and collinear point checks. A successful test, in theory, would mean no discrepancies between the outputs and answers, but to achieve that level of testing I would need to test every possible iteration and position of each shape within the 0 – 100 bounds. Ensuring that level of coverage would be time consuming, so randomly generating them will be as close to that level as I can without it becoming overly verbose.

Consistency Oracle

The purpose of this Oracle is to ensure classification consistency through the comparison of one test execution to another for similarity. My implementation of this involved running a large set of random inputs with no pre-determined exact result. Once the initial run was completed, and the output files were generated, a second run of the same initial data would be run. By comparing two outputs from the same randomized test input, I can ensure that this classifier will consistently generate the same output, given a specified input. This type of test is essential for catching undefined behaviors within the structure of the program that would not be apparent if only deterministic inputs were tested.